COLLABORATIVE PROBLEM SOLVING AND GROUPWARE FOR SOFTWARE DEVELOPMENT

Joanna DeFranco-Tommarello and Fadi P. Deek

When a group is working together toward a common goal, communication and collaboration are key. This is especially true in software development where a project of any significance involves groups of people. Collaborative problem-solving techniques and groupware technology can be a boon to software development because they enhance communication and cooperation. This article examines how software development is affected by collaborative problem solving and decision making, groupware theory and tools, and group cognition and psychology. It also analyzes collaboration tools and their correlation to problem solving and group characteristics.

JOANNA DEFRANCO-TOMMARELLO is a special lecturer for the College of Computing Sciences at the New Jersey Institute of Technology, Newark, New Jersey.

FADI P. DEEK is the associate dean of the College of Computing Sciences at the New Jersey Institute of Technology, Newark, New Jersey, where he is also the director of the information technology program and a professor with the information systems department.

THE OLD ADAGE OF "TWO HEADS ARE better than one" has been realized by many, especially in the software development domain. With the rapid increase of software complexity, applications must be developed as efficiently and timely as possible. Some software can contain millions of lines of code. The objective is to have these large applications developed by several people. Collaboration in the area of software development is therefore a necessity rather than just a benefit of technology.

When developers are geographically dispersed, their communication success may depend on utilizing effective groupware (Nunamaker, 1999). Groupware is any software tool that supports teams whose members work collaboratively on interconnected personal workstations (Zwass, 1998). Groupware systems consists of many components, ranging from the simple to the very complex. These systems can successfully assist a team to work in a shared workspace toward a common goal. Physical space and time may separate team members using such systems. Well-integrated groupware systems promote effective communication, coordination and collaboration within an organization (Aannestad and Hooper, 1997). Groupware and collaborative software development remain relatively new research areas and additional benefits can be achieved. A review of the software engineering literature revealed growing interest in the integration of groupware systems with both the problem solving and software development functionalities. These are addressed later in this article. Next, this article reviews collaborative problem solving and decision making, groupware theory and tools, and group cognition and psychology.

The collaboration of the programmers provided confidence in the solution and enjoyment during the problem-solving process.

COLLABORATIVE PROBLEM-SOLVING AND DECISION-MAKING METHODOLOGY

The contemporary computing professional must work in an environment where programs are thousands or millions of lines long, often extensively modified and maintained rather than constructed, manipulated in a tool-rich environment, where work is usually a team effort, and where the form of a solution has profound impact on future cost and performance (Mulder et al., 1995). These large systems need to be maintained and may have new or changing requirements, and thus require group collaboration.

Wilson et al. (1993) examined collaborative work and its benefit to beginning programmers performing problem-solving and programming tasks and found positive results. The experiment took place with one group of programmers solving a software problem on their own and another set of programmers being allowed to freely speak to a partner. Collaboration enhanced the problem-solving performance of programmers. This study also found that there is evidence that an individual's ability has little effect on team performance. This was explained by stating that the improvement caused by the collaborative effort cancels the effect of any ability lacking in the programmer. There was also evidence that the collaboration of the programmers provided confidence in the solution and enjoyment during the problem-solving process. Collaborative interactions appear to help the beginning programmers analyze and model problems, and may also help them master the analytical skills required by such tasks (Wilson et al., 1993).

In a more recent experiment with experienced software developers (Nosek, 1998), collaboration again was shown to improve the problem-solving process of the team. In this study, all teams outperformed the individual programmers, enjoyed the problem-solving process more, and had greater confidence in their solutions. Another collaborative experiment performed in introductory computer science classes showed that the collaborative learning group had more improvement (pretest to post-test) than did the control group and rated the course somewhat higher (Sabin and Sabin, 1994). These results could suggest that it might be worthwhile to integrate collaborative activities early into the computing curriculum, where problem solving and programming are first taught.

Problem solving is at the heart of software development. While learning to develop software, beginning programmers often encounter difficulty understanding basic problem-solving concepts. Their deficiencies are in problem-solving strategies and tactical knowledge; misconceptions about syntax, semantics, and pragmatics of language constructs; and ineffective pedagogy of programming instruction (Deek and McHugh, 1999). Collaborative problem solving is an activity where groups of two or more people develop a plan for the design of a complex system that will solve an existing problem. A collaborative problem-solving model is a procedure that is followed to facilitate the collaborative problem-solving process. When solving problems as a group, some find the process far more complicated and chaotic than individual problem solving (Finnegan and O'Mahony, 1996). Another barrier to effective collaboration is the existence of those conditions that prevent the free expression of ideas in a group (Hoffman, 1965). For example, conditions listed by Hoffman include participation biases, personal characteristics, and group structure. However, the benefits of collaboration during problem solving far outweigh the process inefficiencies (Hohmann, 1997). Groups appear to be able to deal with complex tasks more effectively than individuals simply because groups have a larger range of skills and abilities (Finnegan and O'Mahony, 1996). Another benefit could be learning, or at least being exposed to the skills and abilities of the other group members (Prey, 1996). For example, learners working in groups need to articulate designs, critiques, and arguments to other group members, thus encouraging the kind of reflection that leads to learning (Guzdial et al., 1996).

One way to facilitate collaborative problem solving, according to Hohmann (1997), is to simply use the same problem-solving methods as used by individuals. Hohmann indicates that it is important for a group to choose and follow a method when problem solving, but it does not have to be a specific method for group problem solving. The group members should also be familiar with the chosen method, thus spending more time solving the problem instead of understanding the problem-solving procedure. "... How a team engages in the use of a (problem solving) method is substantially different from how an individual uses a method" (Hohmann, 1997).

Simon (1997) also described a three-step model for coordination when cooperatively

The means by which the group progresses from problem realization to solution is through the communication of information and ideas, as well as the collaboration of group members.

making decisions: (1) plan development — the plan of behavior is for all the members of the group, not a set of individual plans for each member; (2) communicate plan — the plan needs to be communicated to each member; and (3) behavior modification — the willingness on the part of the individual members to permit their behavior to be guided by the plan. This scheme, according to Simon (1997), relieves each group member of the task of anticipating the behavior of the others as a basis for the member's own. Coordination also enables each group member to adopt the decision made by the group.

Finnegan and O'Mahony (1996) developed a model of group problem solving by observing group decision making and group supporting technology in organizations. While they felt the steps involved in the problem-solving process are important, the means by which the group progresses from problem realization to solution is through the communication of information and ideas, as well as the collaboration of group members. The first stage of the group problem-solving model is problem realization. This stage is a spontaneous process by a specialized group or senior management. They meet on a regular basis to reevaluate the problem in case of any environmental changes. The planning stage is next. This is where the coordination of the sub-groups takes place. The third stage is the "search for information" regarding the problem. The group will discuss the problem in light of the newfound information. Following this discussion is the fourth stage, where the creation of new alternatives is discovered. Next, the evaluation of the alternatives step is performed to prepare for the sixth step. In the sixth stage, an alternative is chosen, followed by selling the alternative to the other groups. Finally, the groups implement the alternative.

Wong (1994) developed a logical, qualitative group problem-solving scheme for making joint decisions and promoting conflict resolution. Cooperative decision making, according to Wong, considers the problem of cooperation; for example, cooperation between software engineers building a complex software application or model. This model consists of three stages: identification, processing, and negotiation. The identification stage is further broken down into three sub-steps. The first step is identifying the decision agenda with priority-ordered important criteria, and the "agent" that is concerned with the criteria (agent is used to describe the system or human working on solving the problem). The next steps are the identification of competing alternatives followed by determining the relationship between some of the competing alternatives. After completing the identification of alternatives comes the processing stage. The purpose of the processing stage is twofold. The first purpose is to develop a set of "preference expressions" for each criterion in the decision agenda determined in the identification stage. Preference expressions are ordering relations for pairs of alternatives. The second part of stage two is to rank order the alternatives, therefore determining the recommended solution. The last stage of the model is the negotiation stage, where the agents negotiate any conflicts that may come into play.

In addition to the theoretical models used in the group problem-solving effort, there are communication issues that must be addressed. Coordinating collaborative activity requires communication among the parties and communication needs to be effective for the problem-solving effort to be successful (Kies et al., 1998). Zhang (1998) addressed group properties and group effectiveness using a purely cognitive perspective. In other words, the research did not only examine the properties of individuals in the group, but also how the individuals interacted with one another. Zhang describes a four-part methodology for group problem solving based on the task distributions across individual representations: consider individual representations as a distributed representation system; decompose the group problem-solving task into individual representations; identify an abstract task structure and its relation to each of the individual representations that were decomposed; and emphasize the interactions among the individual representations.

GROUPWARE THEORY AND TECHNOLOGY

Groupware has changed the way office work is viewed. The globalization of business means that team members are often found at different locations (Nunamaker, 1999). Nunamaker observed that the steady growth in telecommuting and the use of off-site consultants has increased the occurrence of dispersed meetings. The principle functions of groupware, according to Zwass (1998), are information sharing, document authoring, messaging systems, computer conferences, group calendars,

To make groupware effective and efficient in the collaborative software development process, the issues of both collaboration and software development must be understood.

project management, and support for team building.

Groupware systems are generally divided into three categories: synchronous, asynchronous, and a combination of both. Groupware systems that are synchronous run in real-time. A synchronous system can support group communication and collaboration using instant messaging. An example of a synchronous system is an electronic meeting system used for brainstorming sessions. In asynchronous systems, the users read stored messages from other users and are able to store messages for others to view at a later time. An example of an asynchronous system would be an e-mail system. The third category of systems supports both asynchronous and synchronous features.

There have been many comparisons between asynchronous and synchronous groupware systems (Hiltz and Turoff, 1985). Both systems have the benefits associated with increased communication but produce different communication behavior. The key benefit is the obvious one of allowing users in different locations to work together. The variant behaviors include, for example, the tendency in an asynchronous system toward lengthy communications by participants, as well as the discussion of many topics simultaneously (Hiltz and Turoff, 1985). In a synchronous system, on the other hand, participants tend to focus on the current topic.

One model of groupware, developed by Dix (1994), is the Computer Supported Cooperative Work (CSCW) model. The basic components of this model of cooperative or collaborative work are "direct communication" (representing people cooperating or communicating via a computer) and "artifacts" (used either to communicate or accomplish a task and also provide feedback/control to the participants). The objective of this communication is the "understanding" acquired by both participants communicating. Researchers and designers have used this framework to structure both synchronous and asynchronous systems.

Dufner et al. (1999) reported on the social benefits of collaborative learning in an educational environment where asynchronous technologies are used. For example, a benefit would be having time to reflect and think about a problem during asynchronous communication. In addition to reflection, more alternatives could be explored because the asynchronous meetings take place over an extended period of time. Tinzmann (1990) observed several general benefits of collaborative learning in an educational environment. The first benefit is knowledge sharing among teachers and students, as well as peer-oriented knowledge sharing among students, supplemental to the traditional top-down knowledge sharing from the teacher. The collaboration also involves shared authority between teachers and students. Another key characteristic is the significance of a teacher-mediating agent; for example, the teacher can mediate group discussions if the group seems stumped or headed in the wrong direction. Finally, collaboration facilitates heterogeneous grouping, enabling weaker students to learn from the stronger students and vice versa.

One well-known type of groupware system is the Group Decision Support System (GDSS). GDSSs assist group processes, including brainstorming for creative ideas, reaching consensus by voting, surveying experts using the Delphi method, and negotiating, defined as the form of group decision making wherein parties communicate to resolve conflicting interests (Zwass, 1998). GDSS brainstorming entails a group of participants at workstations addressing a problem posed by a manager. All participants generate and post their ideas synchronously. The ideas are then voted upon using the system, which can be accomplished in a matter of minutes. As opposed to a traditional face-to-face meeting, this approach saves time and allows more ideas to be presented in a shorter amount of time; for example, people cannot talk over one another as in a traditional brainstorming meeting. Furthermore, more people can present ideas because self-consciousness is less of an issue in an online environment. The GDSS also allows organized human parallel processing, broader input, equal opportunity for participation, and promotes more discussion than in a face-to-face environment (Zwass, 1998).

Groupware technology should be applied to the decentralization of software development tools (Hahn et al., 1990). To make groupware effective and efficient in the collaborative software development process, the issues of both collaboration and software development must be understood. Hahn et al. (1990) also suggest how cooperative work in software development should be shifted. First, it is important to recognize the development of the group as an organized social process consisting of interactions between the members. Second, the group must be able to negotiate and commit to responsibilities. From this theory, they developed the following four requirements for computer-aided group

Some of the applications reviewed were not exclusively developed to facilitate software development; nonetheless, they are being used for such purposes.

work in software projects. First, support of group interactions must account for human collaboration techniques such as negotiations, commitments, and responsibility contracts. Second, social protocols that underlie group communication must be accounted for in terms of human strategies and policies for argument exchange, contract assignment, decision making, etc. Third, proper tool support and properly controlled tool integration mechanisms must consider domain knowledge of the underlying software project, working procedures, and languages used for specification, design, and implementation. And fourth, all the single modeling efforts must be combined. The entire process of software development is stratified at several layers (requirement analysis, work package planning, programming, etc.), each one covered by a specialized model. These sub-models must be integrated within a composite formal model of software project management to guarantee that transitions between these sub-models are also under formal control. This is important for later replay, discussion, or refutation of reasons for actions on which any rational account of human group work is based.

Groupware is not just for workers who are unable to be in the same area and therefore need to be connected electronically. Groupware can be beneficial for local developers solving problems together because the system can assist in decision making (voting), help keep track of software requirements, and, most importantly, provide means for effective communication. A number of existing tools and environments that support the general groupware features discussed in previous sections are presented next. This selective review is only intended to illustrate some of the variety of available groupware systems.

- Electronic Information Exchange System (EIES), a Web-based computerized conferencing system that supports the activities of a Virtual Classroom™ (Hiltz, 1994; Turoff and Hiltz, 1995), provides asynchronous groupware services such as notifications, class conferencing, electronic mail, and activity functions.
- Virtual-U is a learning environment that integrates conferencing, chat, and gradebook tools. This system provides a framework for designing, delivering, and managing individual courses (Harasim, 1999). It also features e-mail, file exchange, an announcement area, asynchronous discussions and a detailed help system to provide guidelines to course designers.
- Learning Space is an asynchronous learning environment based on Lotus Notes and Notes Server technology. It provides scheduling, a course material database, threaded discussions, user profiles, and provision for user feedback from the discussion manager. Other features of Learning Space include e-mail, an announcement area, file exchange, asynchronous discussions, chat, whiteboard, and video conferencing.
- World Wide Web Course Tool (WebCT) is an interface that facilitates the construction of Web-based courses (Goldberg, 1997). WebCT provides a conferencing system, chat, progress tracking, an announcement area, file exchange, e-mail, timed quizzes, homepage creation, asynchronous discussions, a whiteboard, and search capabilities. This system also facilitates flexibility for designers to modify the look and feel of their courses.
- CoMentor (Hepplestone, 2000) is another system that facilitates online course collaboration. It has synchronous and asynchronous discussion capabilities, e-mail, file exchange databases of previous work, and an announcement board. The goal of this system is to enhance existing courses rather than to provide a complete online course.
- Colloquia, formally known as Learning Landscapes, is a software system that supports group work. Colloquia is distributed, unlike most client/server collaborative systems. The fact that this system is distributed enables users to work offline. This system provides asynchronous group and personal conversation facilities, e-mail, and file exchange.
- TopClass, developed by WBT Systems, is another online learning system that supports training and education. Asynchronous discussion, e-mail, file exchange, and a notice board are all features of this system.

The remaining part of this section focuses on tools that facilitate collaborative problem solving and software development and identifies the collaborative characteristics of each tool. The review of groupware systems for software development will also be selective rather than exhaustive, and is intended only to illustrate some of the variety of available systems. Some of the applications reviewed were not exclusively developed to facilitate software development; nonetheless, they are being used for such purposes.

Conversation Builder is flexible enough to adapt to different processes of software development and other types of collaboration.

The first example of such tools is We-Met. It is is a simple, collaborative graphical editor that allows both asynchronous and synchronous modes of user interaction (Rhyne, 1992). Users work asynchronously, then synchronously broadcast their work to the group. The advantage of having both synchronous and asynchronous features is that a late user can enter a group that has already started, catching up by reviewing messages that occurred before he or she joined. The system is not anonymous; users are explicitly associated with their work, and a history of all work actions is maintained.

Another example is a cooperative design environment called Design Collaboration Support System (DCSS) (Klein, 1994). This system focuses on design conflict detection and resolution, which is a critical component of the cooperative design process. DCSS allows design agents to express design actions, assists in detecting design conflicts, and suggests potential resolutions to the design conflicts detected. This type of conflict resolution is called domain-level conflict, and refers to inconsistencies in a design. It differs from collaborative conflict, where interpersonal issues may be involved.

Computer-Supported Cooperative Training (CSCT) is a synchronous collaborative system designed to enable geographically separated users to work together on a large programming project (Swigger et al., 1995). The primary goal of the system is to allow beginning programmers to collaborate when designing software. Requirement elicitation is the context used to teach collaboration, the objective being to develop a requirements document for a software problem. There are four shared tools: (1) procedural activity, to establish operating procedures via a voting system; (2) problem definition, to specify an agreed on problem statement; (3) criteria establishment, to enter criteria for requirements; and (4) solution activity, to establish a priority for requirements via the voting tool. These tools can be used at any point during the collaboration to identify the software requirements of the problem.

The Karell++ Collaborative Laboratory is an Internet-based, collaborative software development system. This tool has both synchronous and asynchronous collaboration capabilities, enabling users to collaborate in real-time and allowing latecomers to catch up. The goal is to help users develop object-oriented programming techniques by providing a shared development environment for writing programs in the Karell++ language (Rossi, 1999). Users design programs to simulate robots using component-based program elements, and test their results on a graphical simulator.

The Evolving Artifact (EVA) is a collaborative tool that supports software development. Developers use this system to understand problems and develop solutions. This is accomplished by constructing and refining so-called design representations (Ostwald, 1995). This system uses a hypertext environment in which users can view and interact with prototypes and document their comments, the belief being that access to a combination of prototypes and documentation increases problem understanding.

Conversation Builder (CB) is an environment for collaborative work (Kaplan et al., 1992). CB is flexible enough to adapt to different processes of software development and other types of collaboration. The CB environment emphasizes the following characteristics: work activities are collaborative; individuals usually multitask; simultaneous activities are typically dependent on each other; and tasks usually have a number of associated actions to perform. In addition to providing architecture for these characteristics, the system also provides messaging capabilities, software development tools, version management, negotiation activities, shared data modules, and the ability to dynamically interconnect activities.

CoNeX (Hahn et al., 1990) is another example of a tool for collaborative software development. It emphasizes integration of the semantics of the software development domain with aspects of group work, on social strategies to negotiate problems by argumentation, and on assigning responsibilities for task fulfillment by way of contracting. CoNeX contains three tools: (1) the argument editor for negotiating, (2) the contract manager to document dialog, and (3) the conference system for informal messaging. Users can also browse a knowledge base to trace software project history.

To assist in the group problem-solving effort, an asynchronous groupware tool, Web-CCAT, is proposed (Dufner et al., 1999). This tool assists in collaborative work among geographically distributed users. Web-CCAT consists of project management software, computer-aided software engineering (CASE) tools, and GDSS tools. The goal of this tool is to provide a more enriched environment than a face-to-face meeting.

SOLVEIT (Deek, 1997) is a tool that facilitates problem solving in the context of software development. This tool is primarily used

by individual programmers but does support some aspects of collaboration in software development, such as solution integration and component testing. SOLVEIT provides a set of tools that is associated with a six-stage process for problem solving and program development.

The coordination of cognitive processing is a way for the group to have comparable knowledge of the problem area.

GROUP COGNITION AND PSYCHOLOGY

Developers' thought processes are a fundamental area of concern (Stacy and Macmillian, 1995). Cognitive processes are a major attribute of individual problem solving and program development (Deek, 1997). Group problem solving also needs to take into consideration the human cognitive activities involved in problem solving. Cognitive activities within a group are different and more involved than the cognitive activities of one individual solving a problem alone (Hohmann, 1997). This is because one must consider not only the cognitive activities of an individual, but also the cognition activities that result from group interaction.

Stacy and Macmillian (1995) have studied cognitive biases in the area of software engineering. Cognitive biases are the areas where humans consistently and predictably make errors. Their findings suggest three recommendations to eliminate common failure among software engineers. The first is to always opt for empirical investigation over intuition, seek disconfirmatory information, and recast one-sided guidelines as two-sided trade-offs. These recommendations have somewhat of an obvious explanation. Intuition is immediate cognition without the use of rational processes. There is no evidence to back up intuition, so there is more room for error. Therefore, always adopt the option that has empirical support. The second suggestion regarding seeking disconfirmatory information requires the developer to seek information that verifies a situation indirectly. Stacy and Macmillian also suggest that developers ask themselves, "How will I know if the feature doesn't work?" or "How will I know if this isn't the cause of the problem?" And finally, "recast one-sided guidelines as two-sided trade-offs," which implies that a software engineer, when looking at a typical guideline or opinion, should discuss or look at the trade-offs at the same time. This will eliminate biases and controversy from prior experiences. A previous solution may not be valid for the current situation but natural cognition will bring that solution to mind first.

Nosek (1998) explored this research area as well and developed a theory of group cognition. Group cognition is explained as a "combination of distributed and coordinated cognition that directly affects the creation/recreation of distributed and similar knowledge within a team." Nosek also discusses three items needed to create "reasonable knowledge" within a problem-solving group: (1) distributed knowledge, (2) distributed cognition, and (3) coordinated cognitive processing among the group members. Knowledge is defined as the "capacity to act." Cognition is defined as the process of creating knowledge. We can determine the magnitude of these three effects on the collaboration within a group and why these activities need to be coordinated and distributed. Simply stated, the coordination of cognitive processing is a way for the group to have comparable knowledge of the problem area.

Collaboration occurs in many settings that are relevant to this discussion. Kelly and Bostrom (1995) presented a theory regarding a meeting facilitator's interaction with socio-emotional issues in a GSS environment called Adaptive Structuration Theory (AST). This theory suggests that meeting outcomes reflect the manner in which groups appropriate and modify structures present in the meeting process. The three dimensions, in AST, that affect the meeting outcome are (1) faithfulness to the procedures, (2) group attitude, and (3) the group's level of conflict or consensus. Having a meeting facilitator assists in the success of these processes. The facilitator's role is to create a positive environment by appropriately selecting and "facilitating" the use of a structure to match the group's task. Group problem solving can be very similar to meetings, which highlights the importance of discussing these theories in this review.

Somewhat related to group cognition and psychology is the study of human factors. Human factors, in this domain, can be defined as the relationship between humans and their work environment. Human factors is not just a scientific study but also using what is known about the way people really behave and design systems and tools that help make people more productive and happier (Thomas, 1984). Increasing productivity increases the chances of increasing motivation and, in turn, product quality. For the purposes of this discussion, human factors such as ease of use and elements of extrinsic and intrinsic motivation can be added to the group problem-solving model.

When group activities are not properly channeled and coordinated, the interaction among the members could affect the iterative process of solution finding and thus negatively affect the solution.

Forming the team could be almost as important as the problem-solving methodology the team uses during the actual problem solving. No matter how great a collaborative problem-solving model they are using, if the team does not have a complementary set of members, the project may still be unsuccessful. Shneiderman (1980) presents three types of teams: (1) conventional, (2) egoless, and (3) chief programmer. The conventional team assigns one senior member to supervise and direct junior team members. The egoless team focuses on cooperation versus competition. Anything developed is considered the property of the group and not the property of the individual who developed it; therefore, the success or failure of the project is a result of the collaborative effort. The chief programmer team is a team built with defined roles. This is similar to a surgical team where jobs are defined from the outset; for example, surgeon, nurse, and anesthesiologist.

All of the group types have their positive points and their pitfalls. Choosing the team type that is right for a project and team members is an important task and should not be taken lightly. Determination of skill and work ethics, for example, would have to be determined about each team member before the team is formed. Some developers are self-motivated, some are task motivated, and some are interaction motivated. Having too many people in the group who are task-oriented may inhibit the effectiveness of the group's communications (Sommerville, 1996). Having the right personalities composing the group will contribute to the group's cohesiveness.

ANALYSIS

This section provides an analysis of the models and tools presented earlier. The models will be evaluated to determine the kind of collaborative skills and knowledge they require for the tasks of group problem solving. The tools will then be evaluated to examine their support for collaborative problem solving and, where appropriate, their relevance to software development.

Methodology

Finnegan and O'Mahony's (1996) model includes features that benefit and emphasize group activities such as group decision making, but does not provide for activity coordination. An example of the need for coordination is when breaking down the problem into smaller sub-problems and determining which members work on what particular part of the problem. A conflict resolution mechanism is also missing in this model. This would involve the negotiation of activities when determining solution alternatives. Setting specific guidelines for team interaction in collaborative problem solving is important. When group activities are not properly channeled and coordinated, the interaction among the members could affect the iterative process of solution finding and thus negatively affect the solution. Wong's (1994) qualitative group problem-solving model, on the other hand, does focus on conflict resolution. There are definite negotiation attributes to this methodology; however, this method is also lacks a framework for coordination of activities between the group members as well as stresses the iterative process involved in problem solving. Team interaction is not taken into consideration in this model. Hohmann's (1997) theory of collaborative problem solving is one that focuses on communication and collaboration of the process. This method, for example, recognizes that there is a need to account for the fact that group communication changes every time a member is added to the team. However, this model does not provide for an explicit group problem-solving process since Hohmann indicates that the individual problem-solving models can be used. Conflict resolution is also not apparent in this model. Zhang (1998) indicates that collaborative problem solving should not have explicit steps and thus the methodology highlights four steps that must be included when collaboratively solving problems. These steps are similar to what other models already include, such as breaking down the problem into individual representations. However, coordination among group members is also not emphasized here. Additionally, all of the reviewed models emphasize the importance of developing and monitoring the group dynamics. The next section reviews tools.

Technology

We reviewed two types of tools: those that are specific for problem solving and general groupware tools. The evaluation criteria took into consideration the fact that these tools were designed with different types of motivations. All of the problem-solving tools presented provide asynchronous communication capabilities but only some include synchronous communication features. Both modes have their positive

EVA provides suggestions based on a repository of designs.

points. The asynchronous mode allows group members to join a problem-solving session at various times, either by design or necessity, and because of archived entries be able to develop an understanding and familiarity with the progress. Studies have shown that it is beneficial to prevent turn-taking in a collaborative tool, which asynchronous messaging eliminates, because there is a dramatic decrease in performance when one has to "wait his turn" to submit an idea (Prante et al., 2002). Synchronous communication allows real-time discussion, which could expedite the problem-solving process. Of course, having both modes seems more beneficial. We-Met is one tool that supports synchronous interactions but supports only a limited range of collaborative features: meeting discussions, brainstorming, and collaboration archiving. Although it has been used as a problem-solving tool, it does not provide explicit problem-solving facilities. DCSS, on the other hand, provides an interesting problem-solving feature that assists in detecting design conflicts. However, the system does not support conflict resolution among group members, such as might occur during consideration of design alternatives. Like We-Met, DCSS provides no overall problem-solving framework and neither tool was developed to support software development tasks.

Some of the tools reviewed provide specific support for software development. CSCT is a collaborative tool that features synchronous communication, with the main objective being to facilitate requirement elicitation for software development. This tool does not provide any other facilities to support the rest of the software process. In addition, the absence of an asynchronous communication feature prohibits any group member from catching up if he or she enters the process late. Karell++ is a groupware tool that allows both synchronous and asynchronous communication. This is definitely a very beneficial feature for this type of tool. The main objective of this software development tool is teaching object-oriented design concepts. Problem-solving facilities are not included. EVA is an asynchronous group tool that has a primary feature for sharing software prototypes and the development process documentation. Although this tool does not assist in solving problems or developing software, it provides suggestions based on a repository of designs. CB is one system that includes many features for collaboration and supports activities of group software development. There is no particular problem-solving model associated with this application. CoNeX is a very rich environment for software development. This system integrates the semantics of the software development domain with aspects of group work and social strategies to negotiate problems. However, CoNeX only provides limited problem-solving support; that is, it only considers group interactions, not specific methodology. Web-CCAT is a tool that aids in developing software by providing CASE (computer-aided software engineering) tools but is another example of an application that does not focus on the problem-solving component of software development. Summaries of the problem solving and tools reviewed are shown in Table 1. Columns in this table are labeled with the foremost tasks that a group must perform during software development. The rows are labeled with the names of the groupware tools examined. The cells of the table contain the functionality of the tool that facilitates the column heading.

The second set of tools reviewed consists of tools that were designed to accommodate general collaboration. The first is Groove. Groove's features provide a basic framework that assists in the kind of coordinated problem-solving efforts required in software development. However, collaborative software development requires specific problem-solving support, including features or guidelines that enhance group cognitive activities for collaborative problem solving, and explicit guidance and tools for the problem-solving and program-development process, which Groove was not intended to explicitly support. For example, the brainstorming tool provided in Groove allows users to rank ideas, but does not provide an explicit mechanism for voting or other methods or tools for facilitating consensus. Nonetheless, Groove does support group performance, one of the major aspects of the problem-solving process, including ways to identify the tasks of the proposed solution, distributing tasks, and coordinating, communicating, and modifying the solution.

Notes is an application that assists in coordinating a group's efforts but does not provide tools for real-time or asynchronous collaboration activities. The coordinating features only assist in the problem-solving and design efforts of software development. More collaboration tools are necessary to successfully collaborate when problem solving in the area of software development. As previously mentioned, collaborative software development requires specific problem-solving support, including features or

TABLE 1 Summary of Problem-Solving and Software-Development Tools

Tool Name	ID Tasks	Distribution of Tasks	Coordinating Outcomes	Integrating Solutions	Plan Development	Communication Plan
WeMet						Asynchronous and synchronous messaging capabilities
DCSS			Conflict resoulution feature			Asynchronous messaging capabilities
CSCT			Negotiation voting capabilities			Synchronous messaging capabilities
Karell++						Asynchronous and synchronous messaging capabilities
EVA					Contains a database of designs to assist in developing a solution plan	Asynchronous messaging capabilities
CB						Synchronous messaging capabilities
CoNeX			Negotiation voting capabilities		Contains a database of designs to assist in developing a solution plan	Asynchronous and synchronous messaging capabilities
Web-CCAT						Asynchronous messaging capabilities

guidelines that enhance group cognitive activities for collaboratively solving problems. In addition, explicit guidance and tools are needed for the group problem-solving and program-development process, which Lotus Notes R5 was not intended to explicitly support. Additional tools to facilitate activities such as brainstorming, voting, task identification, task distribution, etc. are needed.

GroupSystems is an excellent collaborative tool. Its main goal is to support mission-critical collaborative knowledge activities such as strategic planning and risk assessment. Although this tool was not specifically intended for software development, it has most of the features required to collaboratively solve a software problem. For example, it has a brainstorming tool that is organized to keep the team focused on the brainstorming topic. There are also ample voting tools to make the many group decisions required when solving a problem. Obviously, there is not a step-by-step process to guide the team through specific problem-solving and software-development tasks but all of the tools needed to facilitate each aspect of collaborative problem-solving and software-development process are certainly available in the GroupSystems tool. Comparing it to other systems reviewed, the only features GroupSystems does not have are the chat feature, calendar, Web browsing, and a task list. However, the features contained in GroupSystems have much more of an impact on the collaborative problem-solving and software-development process than the features it is missing.

Requisite Pro is an asynchronous groupware application for requirement management with the exception that only one person at a time can modify the requirement documents. That is, the user is forced to either have exclusive rights to modify entire project or just read-only access. The feature that positively balances the exclusive rights problem is the discussion tool. All users can post messages at any time. Requirement management is a very important task when developing software. Changing requirements during project development has been known to cause unsuccessful results such as "runaway projects." Managing requirements can help minimize these kinds of problems. Requisite Pro is an extensive application to manage project requirements. Using this application as part of a collaborative software development model can only increase its success.

Particularly under the pressure of deadlines, cohesive groups can exhibit cognitive biases that preclude selecting the most effective solution.

The CyberCollaboratory is a good tool to assist in some of the problem-solving and programming tasks of software development. The main focus of the tool is decision support, which is a key task in problem solving but certainly not the only task.

Telelogic's *Doors* product is a requirements management system. Doors is able to capture, link, trace, analyze, and manage information to keep a project compliant with its specific requirements during its life cycle. Doors has multiple tools that give the user multiple ways to access information. This feature benefits the needs of the different roles involved in developing software, such as those of managers, developers, and end users.

Together has several features that enable teams to use it not only for modeling and documentation, but also for actual implementation coding. Features include an editor for multiple languages, a debugger and compiler for Java, code generation, syntax highlighting, auto-indent, etc. Together's integrated debugger enables one to do work in conjunction with the Editor. The debugger also features multithreads, watches, and breakpoints.

WikiWeb is a collaboration tool that operates through a Web browser. The tool is essentially a server that hosts a Web site which can be modified and instantly published. That is, Web pages are automatically created and linked to one another. The main features, in addition to instant Web publishing, are file sharing, page change notifications via e-mail, controlled user access and privileges, page indexing, and full text search.

RUP is a generic Web-based software engineering process that provides a framework to assign and manage tasks and responsibilities within a development organization. RUP attempts to enhance team productivity by delivering what Rational calls "software best practices" to all team members.

We further evaluated these tools based on their ability to enhance or diminish the positive and negative side effects or tendencies that occur when groups of people get together to collaborate. These side effects are natural occurrences that will occur during any type of collaboration. There are existing applications with features that can nurture the positive side effects and reduce the negative side effects. We begin with a discussion describing the side effects highlighted in this article and conclude with a chart showing the correlation between the side effects and the tools reviewed.

Group cohesion is a side effect that is an important phenomenon of group psychology and sociology that must be considered in collaboration, as well as with individual commitment. An example of the correlation between group cohesion and a collaborative application is a brainstorming tool that can help facilitate group cohesion with an environment that provides a way for group members to share their ideas as well as view the ideas of their team members. It will show the group members each individual's commitment. *Distributed learning* is a benefit that has been observed during collaboration (Tinzmann, 1990), where knowledge is shared between the group members. *Consensus building* also occurs during collaboration, where by discussing and sharing the contributions of all the team members, a joint solution will usually result (Constantine, 1990). It is important to build consensus to diminish interpersonal conflict. It has been shown that the impact of interpersonal conflict on systems development is negative and that greater effort toward the prevention of interpersonal conflict needs to occur (Barki and Hartwick, 2001). Another effect of collaboration is *cognitive synchronization*, which occurs when participants make certain they share a common representation of a given subject. *Cognitive overload* (Fussell et al., 1998) is also a possible side effect when a task assigned to a team member is too demanding and not fitting of their skill set. It is important to be aware of every team member's skills so that the team's resources are used efficiently. *Groupthink* (Janis, 1982) occurs when the desire of group members for unanimity overrides their need to evaluate alternatives objectively. Particularly under the pressure of deadlines, cohesive groups can exhibit cognitive biases that preclude selecting the most effective solution. *Cognitive bias* refers to the propensity of individuals to be consistent and predictable in their behavior with respect to the kind of errors they make.

Table 2 shows the correlation between the functionality of the applications reviewed and the side effects that occur during collaboration. For example, the Groove application has brainstorming, chat, document storage, and message board tools that facilitate *group cohesion*.

CONCLUSION

One could infer, based on the benefits of collaboration in general, that collaborative problem solving and software development would

TABLE 2 Tool Functionality versus Collaborative Side Effects

	Group Cohesion	Distributed Learning	Cognitive Bias	Cognitive Sync.	Consensus Building	Group Think	Cognitive Overload
Groove	Brainstorming tool, chat, document storage, message board	Message board, chat	Chat, Internet link tool	Brainstorming tool, link Tool	Message board	Message board, chat	Scheduler, task list
Cyber-Collaboratory	Brainstorming tool, chat, document storage, message board	Message board, chat	Chat, idea organizer	Brainstorming tool	Voting, message board	Message board, chat, idea organizer	
Req Pro							Proj man tool
Doors							Proj man tool
GroupSystems	Brainstorming tool, document storage, message board		Idea organizer	Brainstorming tool	Message board	Idea organizer	Task list
Lotus Notes	Document storage						Scheduler, task list
Together	Document storage						
WikiWeb	Document storage, message board	Message board			Message board	Message board	
RUP	Document storage						

improve the software development process. Collaboration during such activities would allow bringing products to market faster and may also assist software developers in creating solutions to more complex problems with a lower frequency of errors and at a faster rate.

However, this article's review indicates that there is more emphasis on the technology for collaboration and less focus on the methodology. Current models need to consider the psychology and sociology associated with collaborative problem solving. For example, none of the software development groupware tools presented here has taken into consideration such issues as group cognition in collaboration. Jones and Marsh (1997) suggest that this is because a majority of groupware designers are technologists who have both the experience and tools to develop new and effective hardware and software, but these same designers do not have the expertise in social protocols to provide the support necessary in groupware systems. Group cohesion is another aspect of group psychology and sociology that needs to be considered.

Human–computer interaction (HCI) practitioners, whose work combines psychology, social sciences, computer science, and technology, have been addressing such concerns (Carroll, 1997). Shneiderman (1980) indicates that an understanding of human factors, skills, and capacity can improve the design of effective computer systems by applying the techniques and methods of cognitive, social, personnel, and industrial psychology. Shneiderman also indicates that although focusing on psychological and social issues may increase design time and ultimately cost, the design quality will be improved. ▲

References

Aannestad, B. and Hooper, J., "The Future of Groupware in the Interactive Workplace," *HRMagazine*, 12(11), 37–41, November 1997.

Barki, H. and Hartwick, J., "Interpersonal Conflict and Its Management in Information System Development," *MIS Quarterly*, 25(2), 195–228, June 2001.

Brereton, O., Lees, S., Bedson, R., Boldyreff, C., Drummond, S., Layzell, P., Macaulay, L., and Young, R., "Student Collaboration across Universities: A Case Study in Software Engineering," *Thirteenth Conference on Software Education and Training*, March, 2000, 76–86.

Carroll, J., "Human-computer interaction: psychology as a science of design," *International Journal of Human Computer Studies*, 46, 501–522, April 1997.

Constantine, L., "Teamwork Paradigms and the Structured Open Team," *Software Development '90: proceedings of Miller Freeman Publications,* Oakland, CA, 1990, 87–93.

Deek, F.P., An Integrated Environment for Problem Solving and Program Development, unpublished Ph.D. dissertation, New Jersey Institute of Technology, 1997.

Deek, F. P., "The Software Process: A Parallel Approach through Problem Solving and Program Development," *Journal of Computer Science Education,* 9(1), 43–70, April 1999.

Deek, F.P., Hiltz, S.R., Kimmel, H., and Rotter, N., "Cognitive Assessment of Students' Problem Solving and Program Development Skills," *Journal of Engineering Education,* 88(3), 317–326, July 1999.

Deek, F.P. and McHugh, J., " SOLVEIT: An Environment for Problem Solving and Program Development," *Journal of Applied Systems Studies, Special Issue on Distributed Multimedia Systems with Applications,* 2000.

Deek, F.P., McHugh, J., and Hiltz, S.R., "Methodology and Technology for Learning Programming," *Journal of Systems and Information Technology,* 4(1), 25–37, June–July 2000.

Deek, F.P., Turoff, M., and McHugh, J., "A Common Model for Problem Solving and Program Development," *Journal of the IEEE, Transactions on Education,* 42(4), 331–336, November 1999.

Denning, R. and Smith, P., "Teaching Problem-Solving through a Cooperative Learning Environment," *CHI '95 Mosaic of Creativity,* May 1995, 9–10.

Dix, A., Chapter 2 in *Design Issues in CSCW,* edited by Dan Diaper and Colston Sanger, Springer-Verlag London Limited, Great Britain, 1994.

Dufner, D., Kwon, O., and Hadidi, R., "WEB-CCAT: A Collaborative Learning Environment for Geographically Distributed Information Technology Students and Working Professionals," *Communications of the Association for Information Systems,* Vol. 1, Article 12, March 1999, available [online]: http://cais.isworld.org/articles/1-12/article.htm [26 November 2000].

Finnegan, P. and O'Mahony, L., "Group Problem Solving and Decision Making: An Investigation of the Process and the Supporting Technology," *Journal of Information Technology,* 11(3), 211–221, September 1996.

Fussell, S.R., Kraut, R.E., Lerch, F.J., Scherlis, W.L., McNally, M.M., and Cadiz, J.J., "Coordination, Overload and Team Performance: Effects of Team Communication Strategies," CSCW, Seattle Washington, 1998, 275–284.

Goldberg, M., "WebCT and First Year: Student Reaction to and Use of a Web-Based Resource in First Year Computer Science," *ITiCSE '97,* 1997, 127–129.

Guzdial, M., Kolodner, J., Hmelo, C., Narayanan, H., Carlson, D., Rappin, N., Hubscher, R., Turns, J., and Newstetter, W., "Computer Support for the Learning through Complex Problem Solving," *Communications of the ACM,* 39(4) 43–45, 1996.

Hahn, U., Jarke, M., and Rose, T., "Group Work in Software Projects: Integrated Conceptual Models and Collaboration Tools," *Proceedings of the IFIP WG8.4 Conference on Multi-User Interfaces and Applications,* Heraklion, Greece, September 1990, North-Holland, Amsterdam, 83–101.

Harasim, L., "A Framework for Online Learning: The Virtual-U," *IEEE,* September 1999, 44–49.

Hepplestone, S., "coMentor News," December 2000, available [online]: http://comentor.hud.ac.uk, [09 December 2000].

Hiltz, S.R., *The Virtual Classroom: Learning without Limits via Computer Networks,* Ablex Publishing Corp., Norwood, NJ, Human-computer Interaction Series, 1994.

Hiltz, S.R. and Turoff, M., "Structuring Computer-Mediated Communication Systems to Avoid Information Overload," *CACM,* 28(7), 682–689, July 1985.

Hoffman, L.R., "Group Problem Solving" chapter in *Advances in Experimental Social Psychology,* edited by Leonard Berkowitz, Academic Press, New York, 1965.

Hohmann, L., *Journey of the Software Professional,* Prentice Hall PTR, New Jersey, 1997.

Janis, I., *Groupthink: Psychological Studies of Policy Decision,* Houghton, Boston, 1982.

Jarzabek, S. and Huang, R., "The Case for User-Centered CASE Tools," *Communications of the ACM,* 41(8), 93–99, August 1998.

Jones, S. and Marsh, S., "Human–Computer–Human Interaction: Trust in CSCW," *SIGCHI Bulletin,* 29(3), July 1997.

Kaplan, S., Tolone, W., Carroll, A., Bogia, D., and Bignoli, C., "Supporting Collaborative Software Development with ConversationBuilder," *ACM-SDE,* Virginia, pp. 11–20, December 1992.

Kelly, G. and Bostrom, R., "Facilitating the Socio-Emotional Dimension in Group Support Systems Environment," *SIGCPR '95,* Nashville, TN, 1995.

Kies, J., Williges, R., and Rosson, M., "Coordinating Computer-Supported Cooperative Work: A Review of the Research Issues and Strategies," *Journal of the American Society for Information Science,* 49(9), 776–791, 1998.

Klein, M., Chapter 11 in *Design Issues in CSCW,* edited by Dan Diaper and Colston Sanger, Springer-Verlag London Limited, Great Britain, 1994.

McCracken, M. and Waters, R., "WHY? When an Otherwise Successful Intervention Fails," *ITiCSE,* June 1999.

Mulder, M., Haines, J.E., Prey, J.C., and Lidtke, D.K., "Collaborative Learning in Undergraduate Information Science Education," papers of the

26th SISCSE Technical Symposium on Computer Science Education, 1995, 400–401.

Nosek, J., "Augmenting the Social Construction of Knowledge and Artifacts," Air Force Research Laboratory, Report Number AFRL-HE-WP-TR-1998-0082, February 1998.

Nosek, J., "The Case for Collaborative Programming," *Communications of the ACM*, 41(3), 105–108, 1998.

Nunamaker, J., "Collaborative Computing: The Next Millennium," *Computer*, 32(9), 66–71, September 1999.

Ostwald, J., "Supporting Collaborative Design with Representations for Mutual Understanding," *CHI' Companion*, 1995, 69–70.

Prante, T., Magerkurth, C., and Streitz, N., "Developing CSCW Tools for Idea Finding — Empirical Results and Implications for Design," *CSCW '02*, 106–115.

Prey, J.C., "Cooperative Learning and Closed Laboratories in an Undergraduate Computer Science Curriculum," *Proceeding of Integrating Technology into Computer Science Education*, June 1996, Spain, 23–24.

Rhyne, J. and Wolf, C., "Tools for Supporting the Collaborative Process," *Proceedings of the Fifth Annual ACM Symposium on User Interface Software and Technology*, 1992, 161–171.

Rossi, A., "KPPCDL: An Internet Based Shared Environment for Introductory Programming Education," *Proceedings of the 4th Annual SIGCSE/SIGCUE on Innovation and Technology in Computer Science Education*, 1999, 196.

Sabin, R. E. and Sabin, E., "Collaborative Learning in an Introductory Computer Science Course," *SIGCSE Symposium on Computer Science Education*, 1994, 304–308.

Shneiderman, B., *Software Psychology: Human Factors in Computer and Information Systems*, Winthrop Publishers, Inc., Cambridge, MA, 1980.

Simon, H.A., *The New Science of Management*, Harper & Row, New York, 1960.

Simon, H.A., *Administrative Behavior*, fourth edition, The Free Press, New York, 1997.

Sommerville, I., *Software Engineering*, fifth edition, Addison-Wesley, Reading, MA, 1996.

Stacy, W. and Macmillian, J., "Cognitive Bias in Software Engineering," *Communications of the ACM*, 39(6), 57–63, June 1995.

Swigger, K., Brazile, R., and Shin, D., "Teaching Computer Science Students How to Work Together," *CSCL Conference Proceedings*, October 1995, available [online]: http://www-cscl95.indiana.edu/cscl95/swigger.html [26 November 2000].

Thomas, J., *Human Factors and Interactive Computer Systems*, edited by Yannis Vassiliou, Ablex Publishing Corporation, Norwood, NJ, 1984, Chap. 2.

Tinzmann, M.B., Jones, B.F., Fennimore, T.F., Bakker, J., Fine, C., and Pierce, J., "The Collaborative Classroom," *NCREL*, Oak Brook, IL, 1990.

Tinzmann, M.B., Jones, B.F., Fennimore, T.F., Bakker, J., Fine, C., and Pierce, J., "What Is the Collaborative Classroom?," NCREL, Oak Brook, IL, 1990, available [online]: http://trackstar.hprtec.org/main/display.php3?option=frames&track_id=2897 [26 November 2000].

Turoff, M., "Designing a Virtual Classroom," *1995 International Conference on Computer Assisted Instruction*, ICCAI '95.

Turoff, M. and Hiltz, S.R., "Designing and Evaluating a Virtual Classroom," *Journal of Information Technology for Teacher Education*, 4 (2), 197–215, 1995.

Wilson, J., Hoskin, N., and Nosek, J., "The Benefits of Collaboration for Student Programmers," *24th SIGCSE Technical Symposium on Computer Science Education*, February 1993, 160–164.

Wong, Stephen T.C., "Preference-Based Decision Making for Cooperative Knowledge-Based Systems," *ACM Transactions on Information Systems*, 12(4), 407–435, October 1994.

Zhang, J. "A Distributed Representation Approach to Group Problem Solving," *Journal of the American Society for Information Science*, 49(9), 801–809, 1998.

Zwass, V., *Foundations of Information Systems*, Irwin McGraw-Hill, Boston, MA, 1998.